Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (original) A method implementing a native instruction set architecture (ISA) having an emulation engine and an emulated ISA, wherein a fetch engine fetches instructions of the emulated ISA from a memory subsystem, the method comprising:

sending a fetch request signal to the fetch engine;

sending a fetch address to the fetch engine;

retrieving a line of instructions from the memory subsystem, wherein the line of instructions is associated with the fetch address;

storing the fetch address in a fetch address queue;

sending the line of instructions to the emulation engine;

sending a fetch complete signal from the fetch engine to the emulation engine when the line of instructions is sent to the emulation engine, the fetch complete signal being a signal separate from the line of instructions, wherein the step of storing comprises storing the fetch address in the fetch address queue until the fetch complete signal is sent;

storing the fetch address in a queue of the emulation engine, wherein the queue of the emulation engine mirrors the fetch address queue;

progressing the fetch address through pipeline stages of the fetch engine and the emulation engine synchronously using pipeline advance logic;

receiving, at the pipeline advance logic, a delayed version of the fetch complete signal; and

advancing the fetch address through the fetch engine and the emulation engine based on the delayed version of the fetch complete signal.

2. (original) The method of claim 1, wherein a plurality of fetch requests may be processed at the same time.

3. (original) The method of claim 1, further comprising sending the line of instructions to a macroinstruction queue, within the emulation engine.

4. (original) The method of claim 3, further comprising:

determining whether the macroinstruction queue is full or will become full with one or more lines of instructions returning from one or more pending fetch requests; and if the macroinstruction queue is full or will become full with one or more lines of instructions returning from one or more pending fetch requests, waiting until the macroinstruction queue is not full and will not become full to send a new fetch request signal.

5. (previously amended) The method of claim 4, further comprising using a speculative write pointer to determine whether the macroinstruction queue is full or will become full with one or more lines of instructions returning from one or more pending fetch requests by comparing a number of pending fetch requests with a number of open entries in the macroinstruction queue.

6. (original) The method of claim 1, further comprising, if a pending fetch request is canceled due to a pipeline flush,

canceling a pending fetch request; and

clearing the fetch address queue.

7. (original) A method for fetching a line of instructions from a memory subsystem of a mixed architecture CPU into a macroinstruction queue of an emulation engine comprising,

determining whether the macroinstruction queue is full or will become full with one or more lines of instructions returning from one or more pending fetch requests; and

if the macroinstruction queue is not full and will not become full with one or more lines of instructions returning from one or more pending fetch requests,

sending a fetch address to a fetch engine;

retrieving a line of instructions from a memory subsystem into the fetch engine;

sending the line of instructions to the emulation engine;

sending a fetch complete signal from the fetch engine to the emulation engine along with the line of instructions;

if the macroinstruction queue is full or will become full with one or more lines of instructions returning from one or more pending fetch requests, waiting until the macroinstruction queue is not full and will not become full before sending a fetch address to the fetch engine;

storing the fetch address in a queue of the emulation engine, wherein the queue in the emulation engine mirrors the fetch address queue;

progressing the fetch address through pipeline stages of the fetch engine and the emulation engine synchronously using pipeline advance logic;

receiving at the pipeline advance logic a delayed version of the fetch complete signal; and advancing the fetch address through the fetch engine and the emulation engine based on the delayed version of the fetch complete signal.

8. (original) The method of claim 7, further comprising using a speculative write pointer to determine whether the macroinstruction queue is full or will become full with one or more lines of instructions returning from one or more pending fetch requests.

9. (original) The method of claim 8, further comprising, if the macroinstruction queue is not full and will not become full, sending a fetch request signal to the fetch engine along with the fetch address.

10. (original) The method of claim 7, further comprising, if the macroinstruction queue is not full and will not become full, buffering the fetch address in a fetch address queue in the fetch engine.

11. (currently amended) A multi-architecture computer system capable of implementing a native instruction set architecture (ISA) and an emulated ISA, wherein instructions of the native ISA are processed in a native ISA pipeline and instructions of the emulated ISA are processed in an emulated ISA pipeline, the system, comprising:

a memory subsystem of ~~a~~ the native ISA;

a fetch engine of the native ISA, said fetch engine being electrically connected to the memory subsystem of the native ISA wherein the fetch engine accesses the memory subsystem to retrieve a line of instructions from the memory subsystem;

an engine of an emulated ISA, wherein the engine of the emulated ISA is electrically connected to the fetch engine and interfaces with the fetch engine using a handshake protocol, wherein the engine of the emulated ISA receives a line of instructions and a fetch complete signal from the fetch engine; ~~and~~

a fetch address queue that stores a fetch address for the line of instructions retrieved from the memory subsystem when the ~~emulated~~ native ISA pipeline is stalled, wherein the fetch address queue is controlled by the fetch complete signal such that the fetch address is stored in the fetch address queue until the fetch complete signal is received; and

pipeline advance logic, wherein the pipeline advance logic receives a delayed version of the fetch complete signal and advances the fetch address through the fetch engine and the engine of the emulated ISA based on the delayed version of the fetch complete signal.

12. (original) The computer system of claim 11, wherein the engine of the emulated ISA requests the line of instructions and the fetch engine sends the line of instructions to the engine of the emulated ISA.

13. (currently amended) The computer system of claim 11, wherein, if a pending fetch request is canceled due to a pipeline flush,

a pending fetch request signal is canceled; and

~~a~~ the fetch address queue is cleared.

14. (original) The computer system of claim 11, further comprising a macroinstruction queue that receives a line of instructions from the fetch engine.

15. (original) The computer system of claim 14, further comprising a speculative write pointer that prevents the macroinstruction queue from becoming oversubscribed by one or more pending fetch requests, wherein the speculative write pointer may be used to control the sending of a fetch request.